CS 4501-6501 Topics in Cryptography 9 Feb 2018

Lecture 4, Stretching PRGs, Hashing, Chosen Plaintext Security

Lecturer: M. Mahmoody, Scribe: Jingshu Liang, Faysal Hossain Shezan, Mariah Kenny

1 Introduction

Last time we covered computational secrecy, pseudo-random generators and how to use them for encrypting messages longer than the keys in an (single message) indistinguishably-secure way. Let's recall how we encrypted the messages that are twice as long as the key using a pseudo-random generator.

Construction 1.1. Suppose, we want to get an encryption scheme using a PRG g from n bit messages to $2n$ bit messages. Given a key k of length n we can encrypt messages of length $2n$ as follows.

- $g : \{0,1\}^n \leftarrow \{0,1\}^{2n}$
- $\text{Enc}(k, m) = g(k) \oplus m$
- $\text{Dec}(k, c) = g(k) \oplus c$

where $\oplus$ is the bitwise XOR.

Intuition . The intuition is that the ciphertext is indistinguishable from a truly random string of length $2n$ in eyes of computationally bounded adversaries. And as it is indistinguishable, it practically works like a one-time pad. Indeed, since k is chosen randomly, so $g(k)$ will also generate a pseudo random number therefore we can substitute it to a one-time pad encryption where we had a truly random key. Intuition here is once we substitute a pseudo random key in a construction we can assume it to be actually a random number and this is prove able. If one can distinguish two messages of the same length then we can turn one indistinguishable function to the other. (This argument needs to be formalized to be called a proof of security!)

However, note that since the output of the pseudo random generator cannot be truly random, and if the adversary had unbounded time, then they can distinguish the ciphertexts from truly random strings quite effectively.

This is how we constructed the first non-trivial encryption. And now we want to improve this construction in two ways. We want stronger security notions and also want to encrypt more than one message even with the arbitrary length of messages.

Question. Suppose, we want to encrypt one message but it's length is larger than 2n. Here key is still n bit. Let's say the message is of length $3n$. What do we need based on the above encryption algorithm? Answer: If we have a PRG which gives us output of $3n$ bits of length, that would be enough for doing such encryption.

In this lecture we will first go through how to improve pseudo random generator in case of encrypting longer messages more securely, then we will see how to use cryptographic hash function to construct pseudo random generators, and then we will discuss a better notion security that addresses the multi-message aspect.

2 How to make PRGs stretch the output more?

Suppose, we have $g : \{0,1\}^n$ that maps n bits input to $\{0,1\}^{n+1}$, for all $n \in \mathbb{N}$, and that $g(U_n)$ is pseudo random (Namely, for all $t(n) = \text{poly}(n)$-time adversaries A, there is a negligible function $\varepsilon(n)$ such that $g(U_n)$ and U_{n+1} are at most $\varepsilon(n)$ distinguishable by A).

Here, we will do it for just 1 bit of more stretch. But it can be done for n bit stretch as well.

Construction of the new PRG. We will prove that, $G : \{0,1\}^n$ that maps n bits input to $\{0,1\}^{n+2}$ for all n as described below is also a PRG.

1. G takes x of n bits as input, and $x \in \{0,1\}^n$.
2. Then it stretch it to z which is $n+1$ bits and $y \in \{0,1\}^{n+1}$
3. Do the same trick over z and output $G(x) = g(y) = g(g(x))$ which is of length $n+1$. (Note that, if we can exchange pseudo random string with random string, and if it is truly random, then it would happen that if we apply pseudo random generator one more time.)

2.1 Proof of Security for the PRG G

Now the question is whether $z = G(U_n) = g(g(U_n))$ is computationally indistinguishable from U_{n+2}? It is indeed the case. Beyond intuition that this construction works due to pseudo-randomness of z (and that it would behave like a random string), we want to prove this by going through a formal argument that one can verify.

Goal is to start by assuming that $g(U_n)$ is a pseudo random generator, and we want to prove that $G(U_n)$ is also a pseudo random string. We can formally write it as follows: for $g(U_n)$ to be secure $\forall$ polynomial time adversary, A there is some negligible function $\varepsilon(n)$ we can distinguish between $g(U_n) \equiv U_{n+1}$ by at most $\varepsilon(n)$ where $\varepsilon(n)$ is negligible. Namely, We know that,

$$|\Pr[A(g(U_n)) = 1] - \Pr[A(U_{n+1}) = 1]| \leq \varepsilon(n).$$

Now, want to prove that $\forall$ polynomial time adversary B, it holds that $G(U_n) \equiv_{\delta(n)} U_{n+2}$ for some negligible $\delta(.)$, by which we mean that

$$|\Pr[B(G(U_n)) = 1] - \Pr[B(U_{n+2}) = 1]| \leq \delta(n).$$

The next step in the proof is a common proof technique that we will use over and over, and it is called the hybrid argument. We want to show that B cannot distinguish between the real and the ideal worlds by more than $\delta(n)$ advantage for some negligible $\delta(n)$. For that we introduce an intermediate hybrid world in which the adversary B is given $g(U_{n+1})$. Note that there is a reason for going for this hybrid: in the intuition above, we said the key ideas is that $g(U_n)$ looks like U_{n+1}, and this is what we are substituting $g(U_n)$ with.

Real World	Hybrid World	Ideal World
$g(g(U_n)) = G(U_n) \xrightarrow{y} B$ and $B \to b$	$g(g(U_{n+1}) \xrightarrow{y} B$ and $B \to b$	$U_{n+1} \xrightarrow{y} B$ and $B \to b$
$p_R = \Pr[B(y) = 1]$	$p_H = \Pr[B(y) = 1]$	$p_I = \Pr[B(y) = 1]$

We want to prove that $|p_R - p_I| \leq \delta(n)$ for some negligible $\delta(n)$. We will prove the following claims.

Claim 2.1. $|p_H - p_I| \leq \varepsilon(n+1)$.

Proof. This is simply by the definition of g, because in one game we are giving B a uniform U_{n+2} and in the another experiment we are giving it $g(U_{n+1})$. □

Claim 2.2. $|p_R - p_H| \leq \varepsilon(n)$.

Proof. This is the trickier one, but we can use proof by contradiction as follows. If $|p_R - p_H| > \varepsilon(n)$ then we could get a polynomial time adversary, A such that it distinguishes of $g_1(U_n)$ from U_{n+1} by $\varepsilon(n)$ as well. How do we do it? Note that A will be given some $z \in \{0,1\}^{n+1}$, and A wants to build upon the existence of B to do its job! So, $A(z)$ will first apply $g(z) \to y$, and then it gives y to B then outputs $B(y) \to$ (basically whatever B says!). Note that A is now efficiently simulating an experiment for B such that: if A's input is random, B ends up running in the hybrid experiment, and if A's input is pseudorandom, B ends up running in the real experiment. So if B could have made a distinction (which is what we assumed for sake of contraction), A could use it do to its own job! □

Big picture of the argument. More generally, the above idea extends to stitching any random and pseudorandom strings between experiments, as long as the rest of the computation in that experiment is polynomial time (that is how A could simulate the rest!).

Do it yourself. Prove that if $\varepsilon(n)$ is negligible in n, so is $\varepsilon(n+1)$, and thus so is $\varepsilon(n+1) + \varepsilon(n) = \delta(n)$.

3 Cryptographic Hash Functions

Cryptographic Hash Functions are methods that scientists dedicated to using hard puzzles for designing good PRGs. Although they are different from PRGs, they are generally designed to be even stronger, and thus they could be used as PRGs.

Definition 3.1 (Hash Function)**.** A hash function h is any function that can be used to mapping data of arbitrary size, denoted by $x \in \{0,1\}^*$, to data of fixed size (called the digest) denoted by $\{0,1\}^d$, where d is a large constant here.

Note that the input could be of any size, either larger or smaller than the output size, the output is called the digest of the message.

Note that the definition above was only about "completeness" of hashing, and did say nothing about security yet! This is the part that we will be a bit informal for now, but ideally the output should be as randomly looking as possible. In particular, even though it is not a random mapping, intuitively, it should be as close to it as it can, and in particular it should produce a random looking string, (hence being a candidate for being used as a PRG), but we need many more properties from a good hash function.

Closer look at hash functions There are two way to talk about hash functions.

1. Arbitrary input length: $h : \{0,1\}^* \to \{0,1\}^d$ for a constant d.
2. Fixed input length: $h : \{0,1\}^c \to \{0,1\}^d$ for constants d, c.

These two methods are tightly related. The second form serves as a core function to the first form for generating the digest for messages of random length. One of the simplest, yet beautiful construction of extending the second form and generalize it to the first form is called Merkle-Damgrd construction.

Actual constructions. There are many designs for Hash Functions. The famous ones in history are SHA1, SHA2, SHA3. SHA stands for **S***ecure* **H***ashing* **A***lgorithm*. SHA1 generates digests of 160 bits. SHA2 generates can generates 224, 256, 384 or 512 bits. SHA3 can generate digest of arbitrary size. The **Key** property of Cryptographic Hash Functions is that it should be unpredictable as it could be, in particular, it should be **pseudorandom**.

Good hash function should be as random looking as possible. But they should have other good properties as well: for example, it should be difficult to find two different inputs that map to the same digest. (Note that this is hard if the hash function was truly random, then we would need many queries to find a collision - but how many?) It is as if, we get a fresh output every time you feed it with an input (even though there is no randomness involved here, just a deterministic mapping). Since the size of the digest of the message is fixed for most of SHA algorithms, and in general, the input size is larger than the digest length, therefore by pigeon-hole principle we know that there will be collisions, which means that there are more than one input that maps to the same digest. However, since the space of the input is rather huge, finding such a collision pair takes huge amount of time, therefore it won‘t form a practical attack.

There are researchers using both theoretical attacks, as well as various analysis to show the collision in SHA1 algorithm, which makes it no longer a secure Hash Algorithm. There‘s no formal proof of a function to be PRG, as it need to resolve some of the open problems in complexity of computation first. Namely, PRGs imply that $\mathbf{P} \neq \mathbf{NP}$ where $\mathbf{P}$ is the class of polynomial time solvable problems and $\mathbf{NP}$ is the class of *non-deterministic* polynomial time solvable problems, and resolving this problem is a long standing deep open problem in math and computer science.

More Robust Constructions Understanding the minimal mathematical property that leads to pseudo-randomness is important when building the PRG to encrypt messages.

Definition 3.2 (One Way Function)**.** It is a function that maps an input x to output y under such property that it takes polynomial time to map from the input to the output,in other words, $f(x) = y$ can be done in polynomial time. However, for any poly-time adversary A, it should hold that $\mathbf{P}_{y \leftarrow f(U_n)}[A(y) = x' \text{ where } f(x') = y] \leq \epsilon(|x|)$ for some negligible function $\varepsilon(n)$. Note that here we allow the adversary to find *any* pre-image of y, not just demanding it to be x itself.

Onewayness is a weaker definition of PRG in a sense that if g is a PRG, then the same function g shall be one-way function as well (this is a bit tricky to prove, but try to prove it to see where the challenges are!). The fundamental result of Hastad, Impagliazzo, Levin, Luby [**?**] proved the reverse direction also holds: namely, the existence of One Way Function implies the existence of a PRGs. Note that One-wayness seems to be ubiquitous in the universe, such as the chemical reactions is easier in one way but not the other way around. But proving this mathematically for a concrete function g is one of the hardest problem in mathematics! It is technically an open problem, but, in practice, any of the hash functions like SHA256 or SHA3 are more than secure against known attacks.

4 Revisiting One-message Indistinguishability-based Security

There exists a drawback in the indistinguishability-based security that it is tailored to one message, therefore when encrypting the same message, the cipher texts become the same. To formalize it the following way:

$$\forall\ m, m' \in \mathcal{M} = \{0,1\}^n, k \in \{0,1\}^{n/10}, \text{ so that m = m' } \rightarrow Enc(k, m) = Enc(k, m')$$

It indicates that if you encrypt the same message again, the encrypted message C will be the same.

This is called deterministic encryption. Deterministic encryption is not "secure" when we encrypt more than one messages, because it always ends up with the same ciphertext if we encrypt the same thing using the same key again.

Therefore we need to extend the encryption process to be random in a sense that we get refresh output even if we encrypt the same message.

4.1 Necessity for ***Randomized*** Encryption

An Encryption Scheme is composed of three randomized algorithms: a Key Generation algorithm (*Gen*), Encryption algorithm (*Enc*), and Decryption algorithm (*Dec*).

In previous definitions of an Encryption function, the function $\mathrm{Enc}(k, m)$ took in a secret key k and a plaintext message m. The process $\mathrm{Enc}(k, m)$ produces a ciphertext message c.

Definition 4.1 (Randomized Encryption Function)**.** In a randomized encryption function, the function $\mathrm{Enc}(k, m; r)= c$ takes in an additional parameter of randomness r, where $r \in \{0, 1\}^k$ and produces a ciphertext message c. Since encryption is polynomial time, we won't need more than a polynomial number of input bits, so $k \leq \mathrm{poly}(n)$.

Under the previous definition of an encryption function, the production of a ciphertext message is usually denoted as: $\mathrm{Enc}(k, m) = c$. Under the definition of a randomized encryption function, the production of a ciphertext message is denoted as: $c \leftarrow \mathrm{Enc}(k, m)$ to emphasize on the randomness of the process. In this case the "$\leftarrow$" implicitly denotes the randomness of the process.

Recall the completeness condition for a normal encryption function that states: $\mathcal{SKE}$ is *complete* (i.e., it is correct) if for all $k \in \mathcal{K}, m \in \mathcal{M}$, it holds that

$$\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m.$$

To formalize this condition for a randomized encryption function, we have the following:

Definition 4.2 (Randomized Encryption Completeness condition)**.** We say that our encryption scheme is *complete* (i.e., it is correct) if for all $k \in \mathcal{K}, r \in \mathcal{R}$, it holds that

$$\mathrm{Dec}(k, \mathrm{Enc}(k, m; r)) = m.$$

We can even allow a negligible probability error to happen in the completeness condition, but since we can achieve completeness with probability 1, we will usually go with the simpler, better definition, though negligible error is always OK (as it practically does not happen!).

With these new definitions of an Encryption scheme, we will get a new ciphertext message output each time, even if we encrypt the same message. This prevents attackers from being able to differentiate between two encrypted messages being the same versus the two encrypted messages being unique.

5 Security against Chosen-Plaintext Attacks (CPA Security)

Until now we have considered a relatively weak adversary who only passively eavesdrops on the communication between two honest parties. (Of course, our actual definition of the security game allows the adversary to choose the plaintexts that are to be encrypted. Nevertheless, beyond this capability the adversary is completely passive.) In this section, we formally introduce a more powerful type of adversarial attack, called a chosen-plaintext attack (CPA). As it turns out, interestingly, schemes that are secure under these attacks are also automatically secure against multi-message passive attacks.

The basic idea is that the adversary $\mathcal{A}$ is allowed to ask for encryptions of multiple messages. This is formalized by allowing $\mathcal{A}$ to interact freely with an encryption oracle which is viewed as a "black box" that encrypts messages of $\mathcal{A}$'s choice using the secret key k (which is still unknown to $\mathcal{A}$). When $\mathcal{A}$ queries the oracle by providing it with a plaintext m, the oracle returns a ciphertext $c \leftarrow \text{Enc}(k, m_b)$ as the reply using the same key k as that of the challenger and using *fresh* randomness. Namely, if Enc is randomized, the oracle uses fresh random keys each time it answers a query.

The definition of security requires that $\mathcal{A}$ should not be able to distinguish the encryption of two arbitrary messages of the same length, *even when $\mathcal{A}$ is given access to an encryption oracle.*

Construction 5.1 (The CPA indistinguishability experiment $\text{PrivK}^{\mathsf{CPA}}_{\mathcal{A},\Pi}(n)$)**.** :

1. *A key k is generated by running* $\text{Gen}(1^n)$.
2. *The adversary $\mathcal{A}$ is given input 1^n and oracle access to* $\text{Enc}_k(.)$, *and outputs a pair of messages m_0, m_1 of the same length.*
3. *A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow$* $\text{Enc}_k(m_b)$ *is computed and given to $\mathcal{A}$. We call c the* challenge ciphertext.
4. *The adversary $\mathcal{A}$ continues to have oracle access to* $\text{Enc}_k(.)$, *and outputs a bit b'.*
5. *The output of the experiment is defined to be* 1 *if $b' = b$, and* 0 *otherwise. (In case* $\text{PrivK}^{\mathsf{CPA}}_{\mathcal{A},\Pi}(n) = 1$*, we say that $\mathcal{A}$)* succeeded.

Definition 5.2 (Indistinguishable Encryption under CPA attack)**.** A *private-key encryption scheme* $\Pi = (Gen, Enc, Dec)$ has indistinguishable encryptions under a chosen-plaintext attack (*or is* CPA-secure) *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function* negl *such that*

$$\Pr[\text{PrivK}^{\mathsf{CPA}}_{\mathcal{A},\Pi}(n) = 1] \leq 1/2 + \text{negl}(n),$$

where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment.

Any scheme that has indistinguishable encryptions under a chosen-plaintext attack also has indistinguishable encryptions in the presence of an eavesdropper due to the fact that the private-key eavesdropping attack is a special case of private-key CPA attack where the adversary doesn't use its oracle at all.

Guessing vs. indistinguishability. As before, we know that the above definition is equivalent to another on in which we have two experiments/worlds and in the first one the adversary is always given the encryption of m_0 and in the 2nd one it is always given the encryption of m_0 with the goal of outputting a bit that distinguishes the two worlds.

Do it yourself. Prove that CPA security implies the two security notions below. In both of them we will have different security game, and the adversary wants to distinguish between two scenarios.

- Suppose in the security game, the adversary does not have access to the encryption oracle, and suppose the adversary picks two messages m_0, m_1 of the same length, and then the challenger will do either of the following:
  1. In one world, it would encrypt m_0 twice using fresh randomness and send (c, c'), where $c \leftarrow \text{Enc}(k, m_0), c' \leftarrow \text{Enc}(k, m_0)$. (Note that with high probability these two ciphertexts could be different!)
  2. In the other world, it would encrypt m_0 and m_1 independently and send (c, c'), where $c \leftarrow \text{Enc}(k, m_0), c' \leftarrow \text{Enc}(k, m_1)$.
- In the other security game, the adversary will give two pairs of messages (m_0, m'_0) and (m_1, m'_1) all of the same lengths, and then it either receives encryptions of the first pair (in one world) or encryptions of the second pair (in the other world).